Introducción a la Programación

Guía Práctica 7
Funciones sobre listas (tipos complejos)

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Recordar usar las anotaciones de tipado en todas las variables. Por ejemplo: `def función(numero: int) ->bool:`. Las anotaciones de tipado de las colecciones, listas, diccionarios o tuplas, depende de la versión de Python que usamos. Por ejemplo:

- En Python 3.8 son en mayúscula (ej: List, Dict, Tuple) y hay que importar el tipo: `from typing import List, Dict, Tuple`.
- En Python 3.9 son en minúscula (ej: list, dict, tuple) y no hace falta importar nada. Por ejemplo: list[int], list[tuple[int,str]], list[list[int]].

Sugerimos revisar esta página que resumen las formas de uso: `https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html`

En los ejercicios se pueden usar funciones matemáticas como por ejemplo: `sqrt, round, floor, ceil, %`. Ver especificaciones de dichas funciones en la documentación de Python: `https://docs.python.org/es/3.10/library/math.html` y `https://docs.python.org/es/3/library/functions.html`

Para conocer cómo se usan las funciones sobre secuencias, revisar la especificación oficial: `https://docs.python.org/es/3/library/stdtypes.html?highlight=list#typesseq`

1. Recorrido y búsqueda en secuencias

Ejercicio 1. Codificar en Python las siguientes funciones sobre secuencias:

Nota: Cada problema puede tener más de una implementación. Probar utilizando distintas formas de recorrido sobre secuencias, y distintas funciones de Python. No te conformes con una solución, recordar que siempre conviene consultar con tus docentes.

1. `problema pertenece` (in s:$seq\langle\mathbb{Z}\rangle$, in e: $\mathbb{Z}$) : Bool {
   `requiere:` { $True$ }
   `asegura:` { $(res = true) \leftrightarrow$(existe un $i \in \mathbb{Z}$ tal que $0 \leq i < |s| \wedge s[i] = e$)}
   }

   Implementar al menos de 3 formas distintas éste problema.

2. `problema divide_a_todos` (in s:$seq\langle\mathbb{Z}\rangle$, in e: $\mathbb{Z}$) : Bool {
   `requiere:` {$e \neq 0$ }
   `asegura:` { $(res = true) \leftrightarrow$ (para todo $i \in \mathbb{Z}$ si $0 \leq i < |s| \rightarrow s[i]\ mod\ e = 0$)}
   }

3. `problema suma_total` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
   `requiere:` { $True$ }
   `asegura:` { res es la suma de todos los elementos de s}
   }

   Nota: no utilizar la función `sum()` nativa

4. `problema maximo` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
   `requiere:` {$|s| > 0$ }
   `asegura:` { (res = al mayor de todos los números que aparece en s}
   }

Compilado el 2024/07/02 a las 14:47:08

5. `problema minimo` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {

   `requiere:` $\{|s| > 0\}$

   `asegura:` { (res = al menor de todos los números que aparece en s}

   }

6. `problema ordenados` (in s:$seq\langle\mathbb{Z}\rangle$) : Bool {

   `requiere:` { $True$ }

   `asegura:` { $res = true \leftrightarrow$(para todo $i \in \mathbb{Z}$ si $0 \leq i < (|s| - 1) \rightarrow s[i] < s[i+1]$}

   }

7. `problema pos_maximo` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {

   `requiere:` { $True$ }

   `asegura:` { (Si $|s| = 0$, entonces $res = -1$; si no, res = al índice de la posición donde aparece el mayor elemento de s (si hay varios es la primera aparición)}

   }

8. `problema pos_minimo` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {

   `requiere:` { $True$ }

   `asegura:` { (Si $|s| = 0$, entonces $res = -1$; si no, res = al índice de la posición donde aparece el menor elemento de s (si hay varios es la última aparición)}

   }

9. Dada una lista de palabras ($seq\langle seq\langle$Char$\rangle\rangle$), devolver verdadero si alguna palabra tiene longitud mayor a 7. Ejemplo: [“termo”, “gato”, “tener”, “jirafas”], devuelve falso.

10. Dado un texto en formato `string`, devolver verdadero si es palíndromo (se lee igual en ambos sentidos), falso en caso contrario. Las cadenas de texto vacías o con 1 sólo elemento son palíndromo.

11. Recorrer una $seq\langle\mathbb{Z}\rangle$ y devolver verdadero si hay 3 números iguales consecutivos, en cualquier posición y `False` en caso contrario.

12. Recorrer una palabra en formato `string` y devolver `True` si ésta tiene al menos 3 vocales distintas y `False` en caso contrario.

13. Recorrer una $seq\langle\mathbb{Z}\rangle$ y devolver la posición donde inicia la secuencia de números ordenada más larga. Si hay dos subsecuencias de igual longitud devolver la posición donde empieza la primera. La secuencia de entrada es no vacía.

14. Cantidad de dígitos impares.

    `problema cantidad_digitos_impares` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {

    `requiere:` {Todos los elementos de números son mayores o iguales a 0}

    `asegura:` {res es la cantidad total de dígitos impares que aparecen en cada uno de los elementos de números}

    }

    Por ejemplo, si la lista de números es [57, 2383, 812, 246], entonces el resultado esperado sería 5 (los dígitos impares son 5, 7, 3, 3 y 1).

2. Recorrido: filtrando, modificando y procesando secuencias

Ejercicio 2. Implementar las siguientes funciones sobre secuencias pasadas por parámetro:

1. Dada una lista de números, en las posiciones pares borra el valor original y coloca un cero. Esta función modifica el parámetro ingresado, es decir, la lista es un parámetro de tipo *inout*.

2. Lo mismo del punto anterior pero esta vez sin modificar la lista original, devolviendo una nueva lista, igual a la anterior pero con las posiciones pares en cero, es decir, la lista pasada como parámetro es de tipo *in*.

3. Dada una cadena de caracteres devuelva una cadena igual a la anterior, pero sin las vocales. No se agregan espacios, sino que borra la vocal y concatena a continuación.

4. `problema reemplaza_vocales` (in s:$seq\langle$Char$\rangle$) : $seq\langle$Char$\rangle$ {

   `requiere:` { $True$ }

   `asegura:` {$|res| = |s|$}

   `asegura:` {Para todo $i \in \mathbb{Z}$, si $0 \leq i < |res| \rightarrow$ (pertenece(<'a','e','i','o','u'>, s[i]) $\wedge$ res[i] = '_') $\vee$ ($\neg$ pertenece(<'a','e','i','o','u'>, s[i]) $\wedge$ res[i] = s[i])) }

   }

5. `problema da_vuelta_str` (in s:$seq\langle$Char$\rangle$) : $seq\langle$Char$\rangle$ {

   `requiere:` { $True$ }

   `asegura:` {$|res| = |s|$}

   `asegura:` { Para todo $i \in \mathbb{Z}$ si $0 \leq i < |res| \rightarrow res[i] = s[|s| - i - 1]$}

   }

6. `problema eliminar_repetidos` (in s:$seq\langle$Char$\rangle$) : $seq\langle$Char$\rangle$ {

   `requiere:` { $True$ }

   `asegura:` {($|res| \leq |s|$) $\wedge$ (para todo $i \in \mathbb{Z}$ si $0 \leq i < |s| \rightarrow pertenece(s[i], res)$) $\wedge$ (para todo $i, j \in \mathbb{Z}$ si $(0 \leq i, j < |res| \wedge i \neq j) \rightarrow res[i] \neq res[j]$)}

   }

Ejercicio 3. Implementar una función para conocer el estado de aprobación de una materia a partir de las notas obtenidas por un/a alumno/a cumpliendo con la siguiente especificación:

`problema aprobado` (in notas: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
`requiere:` {$|notas| > 0$}
`requiere:` {Para todo $i \in \mathbb{Z}$ si $0 \leq i < |notas| \rightarrow 0 \leq notas[i] \leq 10$)}
`asegura:` {$res = 1 \leftrightarrow$ todos los elementos de $notas$ son mayores o iguales a 4 y el promedio es mayor o igual a 7}
`asegura:` {$res = 2 \leftrightarrow$ todos los elementos de $notas$ son mayores o iguales a 4 y el promedio está entre 4 (inclusive) y 7}
`asegura:` {$res = 3 \leftrightarrow$ alguno de los elementos de $notas$ es menor a 4 o el promedio es menor a 4}
}

Ejercicio 4. Dada una lista de tuplas, que representa un historial de movimientos en una cuenta bancaria, devolver el saldo actual. Asumir que el saldo inicial es 0. Las tuplas tienen una letra que nos indica el tipo de movimiento "I" para ingreso de dinero y "R" para retiro de dinero, y además el monto de cada operación. Por ejemplo, si la lista de tuplas es `[(''I'', 2000), (''R'', 20),(''R'', 1000),(''I'', 300)]` entonces el saldo actual es 1280.

3. Matrices

Ejercicio 5. Analizando parámetros in y out vs. resultado:

1. `problema pertenece_a_cada_uno_version_1` (in s:$seq\langle seq\langle\mathbb{Z}\rangle\rangle$, in e:$\mathbb{Z}$, out res: $seq\langle$Bool$\rangle$) {

   `requiere:` { $True$ }

   `asegura:` { $|res| \geq |s|$}

   `asegura:` { Para todo $i \in \mathbb{Z}$ si $0 \leq i < |s| \rightarrow (res[i] = true \leftrightarrow pertenece(s[i], e))$}

   }

   Nota: Reutilizar la función `pertenece()` implementada previamente para listas.

2. `problema pertenece_a_cada_uno_version_2` (in s:$seq\langle seq\langle\mathbb{Z}\rangle\rangle$, in e:$\mathbb{Z}$, out res: $seq\langle$Bool$\rangle$) {

   `requiere:` { $True$ }

   `asegura:` {$|res| = |s|$}

   `asegura:` { Para todo $i \in \mathbb{Z}$ si $0 \leq i < |s| \rightarrow (res[i] = true \leftrightarrow pertenece(s[i], e))$}

   }

3. `problema pertenece_a_cada_uno_version_3` (in s:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$, in e:$\mathbb{Z}$) : $seq\langle$Bool$\rangle$ {

   `requiere:` { $True$ }

   `asegura:` { $|res| = |s|$}

   `asegura:` { Para todo $i \in \mathbb{Z}$ si $0 \leq i < |s| \rightarrow (res[i] = true \leftrightarrow pertenece(s[i], e))$}

   }

4. **Pensar:** ¿Cómo cambia este problema respecto de la versión 1? Pensar en relación de fuerza entre: implementación en Python y las especificaciones. ¿Se puede usar la implementación del ejercicio 2 para la especificación del 1? ¿Se puede usar la implementación del ejercicio 1 para la especificación del 2? Justificar su respuesta.

Ejercicio 6. Implementar las siguientes funciones sobre matrices (secuencias de secuencias):

1. `problema es_matriz` (in s:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : Bool {

   `requiere:` { $True$ }

   `asegura:` { $res = true \leftrightarrow (|s| > 0) \wedge (|s[0]| > 0) \wedge$ (Para todo $i \in \mathbb{Z}$ si $0 \leq i < |s| \rightarrow |s[i]| = |s[0]|$)}

   }

2. `problema filas_ordenadas` (in m:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$, out res: $seq\langle$Bool$\rangle$) {

   `requiere:` { $esMatriz(m)$}

   `asegura:` { Para todo $i \in \mathbb{Z}$ si $0 \leq i < |res| \rightarrow (res[i] = true \leftrightarrow ordenados(s[i]))$ }

   }

   Nota: Reutilizar la función `ordenados()` implementada previamente para listas

3. `problema columna` (in m:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$, in c: $\mathbb{Z}$) : $seq\langle \mathbb{Z}\rangle$ {

   `requiere:` { $esMatriz(m)$}

   `requiere:` { $c < |m[0]|$}

   `requiere:` { $c \geq 0$}

   `asegura:` { Devuelve una secuencia con exactamente los mismos elementos de la columna c de la matriz m, en el mismo orden que aparecen}

   }

4. `problema columnas_ordenadas` (in m:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : $seq\langle$Bool$\rangle$ {

   `requiere:` { $esMatriz(m)$}

   `asegura:` { Para toda columna $c \in m \rightarrow (res[c] = true \leftrightarrow ordenados(columna(m, c)))$ }

   }

   Nota: Reutilizar la función `ordenados()` implementada previamente para listas

5. `problema transponer` (in m:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : $seq\langle seq\langle \mathbb{Z}\rangle\rangle$ {

   `requiere:` { $esMatriz(m)$}

   `asegura:` {Devuelve m^t (o sea la matriz transpuesta)}

   }

   Nota: Usar `columna()` para ir obteniendo todas las columnas de la matriz.

6. Ta-Te-Ti Tradicional:

   `problema quien_gana_tateti` (in m:$seq\langle seq\langle$Char$\rangle\rangle$) : $\mathbb{Z}$ {

   `requiere:` {esMatriz(m)}

   `requiere:` {$|m| = 3$}

   `requiere:` {$|m[0]| = 3$}

   `requiere:` {En la matriz si hay 3 X alineadas verticalmente $\implies$ no hay 3 O alineadas verticalmente}

   `requiere:` {En la matriz si hay 3 O alineadas verticalmente $\implies$ no hay 3 X alineadas verticalmente}

`requiere:` {En la matriz si hay 3 X alineadas horizontalmente $\implies$ no hay 3 O alineadas horizontalmente}

`requiere:` {En la matriz si hay 3 O alineadas horizontalmente $\implies$ no hay 3 X alineadas horizontalmente}

`requiere:` {(Para todo i,j $\in \{0, 1, 2\}$) (m[i][j] = $X \vee$ m[i][j] = $O \vee$ m[i][j] = " ")}

`asegura:` {Si hay 3 O alineadas verticalmente, horizontalmente o en diagonal, devuelve 0}

`asegura:` {Si hay 3 X alineadas verticalmente, horizontalmente o en diagonal, devuelve 1}

`asegura:` {Si no hay ni 3 X, ni 3 O alineadas verticalmente, horizontalmente o en diagonal, devuelve 2}

}

7. **Opcional:** Implementar una función que tome un entero d y otro p y eleve una matriz cuadrada de tamaño d con valores generados al azar a la potencia p. Es decir, multiplique a la matriz generada al azar por sí misma p veces. Realizar experimentos con diferentes valores de d. ¿Qué pasa con valores muy grandes?

   Nota 1: recordá que en la multiplicación de una matriz cuadrada de dimensión d por si misma cada posición se calcula como res[i][j] = $\sum_{k=0}^{d-1}(m[i][k] \times m[k][j])$

   Nota 2: para generar una matriz cuadrada de dimensión d con valores aleatorios hay muchas opciones de implementación, analizar las siguientes usando la biblioteca numpy (ver recuadro):

   Opción 1:

   ```
   import numpy as np
   m = np.random.random((d, d))
   ```
   [1]

   Opción 2:

   ```
   import numpy as np
   m = np.random.randint(i,f, (d, d))
   ```
   [2]

   > Para poder importar la biblioteca `numpy` es necesario instalarla primero. Y para ello es necesario tener instalado un gestor de paquetes, por ejemplo `pip3` (**Ubuntu:** `sudo apt install pip3`. **Windows:** se instala junto con Python). Una vez instalado `pip3` se ejecuta `pip3 install numpy`.

4. Programas interactivos usando secuencias

Ejercicio 7. Vamos a elaborar programas interactivos (usando la función `input()`[3]) que nos permita solicitar al usuario información cuando usamos las funciones.

1. Implementar una función para construir una lista con los nombres de mis estudiantes. La función solicitará al usuario los nombres hasta que ingrese la palabra "listo", o vacío (el usuario aprieta ENTER sin escribir nada). Devuelve la lista con todos los nombres ingresados.

2. Implementar una función que devuelve una lista con el historial de un monedero electrónico (por ejemplo la SUBE). El usuario debe seleccionar en cada paso si quiere:

   - "C" = Cargar créditos,
   - "D" = Descontar créditos,
   - "X" = Finalizar la simulación (terminar el programa).

   En los casos de cargar y descontar créditos, el programa debe además solicitar el monto para la operación. Vamos a asumir que el monedero comienza en cero. Para guardar la información grabaremos en el historial tuplas que representen los casos de cargar ("C", monto a cargar) y descontar crédito ("D", monto a descontar).

3. Vamos a escribir un programa para simular el juego conocido como 7 y medio. El mismo deberá generar un número aleatorio entre 0 y 12 (excluyendo el 8 y 9) y deberá luego preguntarle al usuario si desea seguir sacando otra "carta" o plantarse. En este último caso el programa debe terminar. Los números aleatorios obtenidos deberán sumarse según el número obtenido salvo por las "figuras" (10, 11 y 12) que sumarán medio punto cada una. El programa debe ir acumulando los valores y si se pasa de 7.5 debe informar que el usuario ha perdido. Al finalizar la función devuelve el historial de "cartas" que hizo que el usuario gane o pierda. Para generar números pseudo-aleatorios entre 1 y 12 utilizaremos la función `random.randint(1,12)`. Al mismo tiempo, la función `random.choice()` puede ser de gran ayuda a la hora de repartir cartas.

[1] `https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.random.html#numpy.random.Generator.random`
[2] `https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html`
[3] `https://docs.python.org/es/3/library/functions.html?highlight=input#input`

4. Analizar la fortaleza de una contraseña. Solicitar al usuario que ingrese un texto que será su contraseña. Armar una función que tenga de parámetro de entrada un `string` con la contraseña a analizar, y la salida otro `string` con tres posibles valores: VERDE, AMARILLA y ROJA. **Nota:** en python la "ñ/Ñ" es considerado un carácter especial y no se comporta como cualquier otra letra. String es $seq\langle \mathsf{Char} \rangle$. **Consejo:** para ver si una letra es mayúscula se puede ver si está ordenada entre A y Z.

   - La contraseña será VERDE si:
     a) la longitud es mayor a 8 caracteres
     b) tiene al menos 1 letra minúscula.
     c) tiene al menos 1 letra mayúscula.
     d) tiene al menos 1 dígito numérico (0..9)
   - La contraseña será ROJA si:
     a) la longitud es menor a 5 caracteres.
   - En caso contrario será AMARILLA.